Document Creation

Contributing to Play Thanks for the interest in contributing to Play! Contributing Etiquette ■ Please see the Contributor Code of Conduct for information on the rules of conduct. Creating a Document ■ Locate or create the *.md file in the component's directory. Modify the documentation. To update any of the auto generated documentation, make the relevant changes in the following places: Usage : update the component's usage examples in the component's usage/ directory Building Changes ■ Once all changes have been made and the documentation has been updated, run npm run build . Review the changes and, if everything looks correct, commit the changes. Make sure the build has finished before committing. If you made changes to the documentation, properties, methods, or anything else that requires an update to a generate file, this needs to be committed. After the changes have been pushed, publish the branch and create a pull request . Creating a Pull Request ■ We appreciate you taking the time to contribute! Before submitting a pull request, we ask that you please create a document that explains the document and let us know that you plan on creating a pull request for it. If a PR already exists, please comment on that PR letting us know you would like to submit a pull request for it. This helps us to keep track of the pull request and make sure there isn't duplicated effort. Submit Pull Request ■ Create a new pull request with the master branch as the base . You may need to click on compare across forks to find your changes. See the Creating a pull request from a fork GitHub help article for more information. Please fill out the provided Pull Request template to the best of your ability and include any issues that are related. Commit Message Guidelines ■ We have very precise rules over how our git commit messages should be formatted. This leads to readable messages that are easy to follow when looking through the project history. Our format closely resembles Angular's commit message guidelines . Commit Message Format ■ We follow the Conventional Commits specification . A commit message consists of a header , body and footer . The header has a type , scope and subject : <type>(<scope>): <subject> <BLANK LINE> <body> <BLANK LINE> <footer> The header is mandatory and the scope of the header is optional. Revert ■ If the commit reverts a previous commit, it should begin with revert: , followed by the header of the reverted commit. In the body it should say: This reverts commit <hash>. , where the hash is the SHA of the commit being reverted. Type ■ If the prefix is feat , fix or perf , it will appear in the changelog. However if there is any BREAKING CHANGE , the commit will always appear in the changelog. Must be one of the following: feat : A new feature fix : A bug fix docs : Documentation only changes style : Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc) refactor : A code change that neither fixes a bug nor adds a feature perf : A code change that improves performance test : Adding missing tests chore : Changes to the build process or auxiliary tools and libraries such as documentation generation Scope ■ The scope can be anything specifying place of the commit change. Usually it will refer to a component but it can also refer to a utility. For example action-sheet , button , css , menu , nav , etc. If you

make multiple commits for the same component, please keep the naming of this component consistent. For example, if you make a change to navigation and the first commit is fix(nav) , you should continue to use nav for any more commits related to navigation. As a general rule, if you're modifying a component use the name of the folder. Subject ■ The subject contains succinct description of the change: use the imperative, present tense: "change" not "changed" nor "changes" do not capitalize first letter do not place a period . at the end entire length of the commit message must not go over 50 characters describe what the commit does, not what issue it relates to or fixes be brief, yet descriptive - we should have a good understanding of what the commit does by reading the subject Body ■ Just as in the subject , use the imperative, present tense: "change" not "changed" nor "changes".
The body should include the motivation for the change and contrast this with previous behavior. Footer ■ The footer should contain any information about Breaking Changes and is also the place to
reference GitHub issues that this commit Closes . Breaking Changes should start with the word BREAKING CHANGE: with a space or two newlines. The rest of the commit message is then used for this. Examples ■ Does not appear in the generated changelog: docs(changelog): update steps to update Appears under "Features" header, toast subheader: feat(toast): add 'buttons' property Appears under "Bug Fixes" header, skeleton-text subheader, with a link to issue #28: fix(skeleton-text): use proper color when animated closes #28 Appears under "Performance Improvements" header, and under "Breaking Changes" with the breaking change explanation: perf(css): remove all css utility attributes BREAKING CHANGE: The CSS utility attributes have been removed. Use CSS classes instead. Appears under "Breaking Changes" with the breaking change explanation: refactor(animations): update to new animation system BREAKING CHANGE: Removes the old animation system to use the new animations. The following commit and commit 667ecc1 do not appear in the changelog if they are under the same release. If not, the revert commit appears under the "Reverts" header. revert: feat(skeleton-text): add animated property This reverts commit 667ecc1654a317a13331b17617d973392f415f02.